Mount()

Vulnerable to TOCTOU issues

Sean Barnum, Cigital, Inc. [vita[1]]

2007-04-02

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 9284 bytes

Attack Category	• Path spoofing or confusion problem
Vulnerability Category	• Indeterminate File/Path • TOCTOU - Time of Check, Time of Use
Software Context	• File Path Management
Location	• sys/mount.h
Description	Mount() is used to mount filesystems or directory structures to a specified directory. Because mount must be run as a superuser, there are inherent security concerns such as time-of-check, time-of-use (TOCTOU). Any programs using mount() should be well scrutinized and run with the lowest privileges possible. It will operate on symbolic links which further exacerbates the problem. The risks here are that an attacker could mount a filesystem that was not intended to be mounted which could possibly lead to a disclosure or integrity violation of sensitive information.
APIs	**Function Name** / **Comments** mount() /
Method of Attack	The key issue with respect to TOCTOU vulnerabilities is that programs make assumptions about atomicity of actions. It is assumed that checking the state or identity of a targeted resource followed by an action on that resource is all one action. In reality, there is a period of time between the check and the use that allows either an attacker to intentionally or another interleaved process or thread to unintentionally change the state of the targeted resource and yield unexpected and undesired results. An attacker would take a program that uses mount() insecurely (that is, it operates on a relative path or an absolute path (which could be a symbolic link) that the attacker can control) and mount a filesystem

1. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/35-BSI.html (Barnum, Sean)

	or directory of his or here choice, possibly to a target of his or her choice. This rule focuses on the vulnerability of mounting an arbitrary source to a target but it easily could be a fixed source that is mounted to an arbitrary target.
Exception Criteria	Iif mount is run on absolute file paths are that are not in control of the currently running user, this problem should be mitigated.

Solutions

Solution Applicability	Solution Description	Solution Efficacy
This solution is applicable if the application can be adapted to use absolute file paths, can check the access on these file paths, and can create symbolic and hard links in place of using mount.	Use hard links and symbolic links when possible to graft one directory structure to another; mount is not necessary. Do not mount sources user-specified sources or those that the current user has control over (i.e. symbolic links, relative file paths) and always mount absolute file paths.	This solution will reduce the liklihood of a program using mount() being tricked into mounting an unintended file-system.
This solution is applicable if portions of the program can be run without super-user access.	Set the effective user ID (euid) and group id (egid) to that of the real user except when mount() needs to be performed.	This will reduce the exposure of the application to abuses of the super-user privilege by only using it when absolutely necessary.
Generally applicable.	The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the	Does not resolve the underlying vulnerability but limits the false sense of security given by the check.

		underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.	
	Generally applicable.	Limit the interleaving of operations on resources from multiple processes.	Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.
	Generally applicable.	Limit the spread of time (cycles) between the check and use of a resource.	Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.
	Generally applicable.	Recheck the resource after the use call to verify that the action was taken appropriately.	Effective in some cases.

Signature Details	int mount(const char *source, const char *target , const char *filesystemtype, unsigned long mountflags , const void *data);

Examples of Incorrect Code	`/* Improper use of mount on a relative path (and it could be a symbolic link!)`
	`* fs_type is set to the filesystem type to be expected`
	`* mount_flags is set to the flags to be used in this situation`
	`* mount_params is set to the parameters for the specific file system`
	`*/`

```
mount(~/source, /mnt/
target, fs_type, mount_flags,
mount_params);
```

```
/* Improper use of mount on an
absolute file path that the user
controls
* fs_type is set to the
filesystem type to be expected
* mount_flags is set to the flags
to be used in this situation
* mount_params is set to the
parameters for the specific file
system
*/
mount(/home/current_user_name/
source, /mnt/target, fs_type,
mount_flags, mount_params);
```

```
/* Improper use of super-user
privileges to write to a file in
the current directory (relative)
* The buffer data and it's
length, data_len, have already
been specified.
*/
FILE *fp = fopen("log.txt", "w");
fwrite(data, 1, data_len, fp);
fclose(fp);

/* fs_type is set to the
filesystem type to be expected
* mount_flags is set to the flags
to be used in this situation
* mount_params is set to the
parameters for the specific file
system
*/
mount(~/source, /mnt/
target, fs_type, mount_flags,
mount_params);
```

Examples of Corrected Code

```
/* Proper use of the mount
command:
* We will illustrate dropping
privileges until we need our
super-user privileges and the
proper specification of the mount
command.*/

//Get the effective user of the
running process. This will be the
program's user or group owner if
setuid or setgid is used.
uid_t init_uid = geteuid();
```

```
gid_t init_gid = getegid();

//Drop to the privileges of the
user who is runnig the process.
seteuid(getuid());
setegid(getgid());

//Do unprivileged tasks...

//Jump back up to a privileged
effective user
seteuid(init_uid);
setegid(init_gid);

/* A absolute path to the source
is specified and it is not
controlled by the user (at least
not by default on most modern
linux systems.
* The filesystem type is
specified. If an attacker were to
try to mount a file system that
was of a different type, it would
fail.
* The filesystem is loaded with
very restrictive permissions.
*/
if (mount(/dev/hda1, /mnt/target,
"ext3", MS_NOEXEC | MS_NOSUID |
MS_RDONLY, NULL) < 0)
return -1; //Return -1 on error.

//Drop to the privileges of the
user who is runnig the process.
seteuid(getuid());
setegid(getgid());

//Do more unprivileged tasks.
```

Source References	• ITS4 Source Code Vulnerability Scanning Tool [2] • A vague reference [3] • The C mount() functions [4]
Recommended Resource	
Discriminant Set	**Operating System**: • UNIX (All) **Languages**: • C • C++

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com